Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

An Approach Towards High Productivity Computing

Zorislav Shoyat

Centre for Information and Computer Science
Rudjer Boshkovich Institute, Zagreb, Croatia
sojat@irb.hr

Abstract

The notion of what exactly we mean by productivity is largely depending on the active paradigms of a particular field and, on a global level, on the present prevailing social, cultural, scientific and spiritual paradigms and environment. It follows that in a long term any specific definition of productivity will have to be changed. Unfortunately, due to the historical processes, present day human-computer communication is on an extremely low level of language complexity. Consequently our present day productivity in using computers from the idea till the implementation is very low. This is primarily due to the circulus vitiosus of interdependecy of (hardware) computer architectures and popular computer programming languages based on the designs of the first Electronic Brains of the mid-last century. The natural, human Language is the prime Human tool for building a common model of the Universe, a huge fractal dynamic system, i.e. machine, whose sub-machines are smaller fractal machines consisting of a series which goes through dialects, sociolects down to idiolects. On the other hand, regarding strictly formal non-adaptable "programming" languages we see that almost all our computer linguistic efforts are oriented towards fixed expressions which are simple enough to be easily and efficiently translated into the scalar serial presently prevailing computer architecture(s). Therefore a new, fresh approach is proposed, based on the idea that the lowest possible level of a computer system shall understand a natural-like communication language, which is contextful and deals with Information, not with Data without Meta-Data. By significantly leveling up the human-computer interaction towards the ideals of a semi-natural language completely new approaches for High Productivity Computing, both on Hardware and on Software level can be thought out, and the NESUS WG1 Focus Group High Productivity Computing has been established, to historically, futuristically and realistically define and, based on that, develop, through partner collaboration projects, such a (possible) High Productivity System based on specific hardware and software.

Keywords High Productivity Computing, Computer History, Human Computers, Natural Language, Programming Languages, Productivity, Data, Information, Data Processing, Information Processing, Focus Group High Productivity Computing, NESUS

I. What is Productivity?

The word *productivity* comes from the Latin words *pro* ("for") and *duco, 3.* ("to lead"). In this sense pro-duco means to lead towards some thing, and has generally the meaning of something which was made by a process aimed towards that result. Productivity is therefore actually lively occasioning of making something, and encompasses, in its general sense, the whole path to be followed from an idea to its realisation. Therefore we can speak of productivity in sciences, arts, technologies, and even sports.

Naturally, through constant change in the Human society, and the constant change of particular techniques being employed within each, old or newly developed, field of human enterprise, particular aspects of the term productivity are being differently emphasized. The standpoint from which we regard what actually productivity is as applied to a specific (sub-)field in a specific moment in time is directly depending on the present and projected future needs inside that particular field, as well as the present and projected needs of the society in which this field is embedded.

Therefore it is obvious that the notion of what exactly we mean by *productivity* is largely depending on the present state of affairs of the field we are applying the notion to, and, on a more global level, on the present prevailing social, cultural, scientific and spiritual paradigm and environment. It follows that in a long term any specific definition of productivity will have to be changed. However the general definition, productivity being the lively flowing over a process path from an idea to its realisation, holds in all cases. The length of the process path, be it in time, material, effort..., is the generic measure of productivity, a long path (highly time-consuming realisation, enormous amount of effort...) shows low productivity, a short path high productivity.

From this generic definition we can consequently easily adapt the specific notions of productivity in specific fields of human endeavour, according to the mentioned field and civilisation paradigms, standpoints and preferences. It is important to note that the notion of productivity in all of its wide semantic field is directly connected with the notion of *technique*. The use of a certain technique applied to certain processes will consequently directly induce the productivity of that application. A *technique* is actually the way in which a process (from inception to realisation of whatever is produced) is performed. A good technique is the 'tool' to achieve a short path, i.e. high productivity. Technology is therefore a field aiming towards high productivity by rationally organising applications of different techniques to processes.

II. Productivity and Computers

Once upon a time we had simple calculators, abaci and similar, and writing equipment (pen and paper, stylus and papirus, chisel and

stone...). Therefore to make any more complicated calculations a computer, a person who knows how to calculate and compute, was needed. A Computator was known already in ancient Rome, and the first mention of people being Computers in English dates back to 1613. There were quite a lot of areas computers were employed in - ranging from purely scientific applications through navigation, commerce, balistics, finances, building, designing... to, and in the previous, 20^{th} century, even mostly, war efforts.

And, as sad as it is, exactly those war efforts of the last century were necessitating the employment of a huge number of computers, mostly women, to raise the productivity primarily of enemy code breaking and balistic calculations. But we have already seen in the history of technology that certain repetitive operations, exactly programmed, can be more productive when using mechanical means (as for example the programmed looms) or very entertaining (as e.g. the roll-piano). So, accumbent on the existing developments, the programmed loom/roll-piano, the Hollerith sorting machines, mechanical calculators, Babbage's machine constructions, the work of mrs. Ada Lovelace (and not to forget the Zuse mechanical computers), and naturally abaci, during the Second World War we started developing electromechanical, electrical and electronic equipment which could take over the most tedious parts of the jobs computers had to do, and therefore significantly raise the overall productivity of whatever they had to compute.

II.1 Initial Aim of Computers

It is hard exactly to know if the initial aim of non-human computers (or, better to say, calculating and computing equipment) was primarily to take off the burden of tedious repetitive operations in which humans make a lot of errors, or to shorten the time individual calculations in a process of computing something take, or both of those (which actually seems most probable). Anyway, the development of those first non-human computers radically changed the world in that instant (as strange as it sounds, but here we will not go into the philosophycal justification of this statement).

Regarding the productivity, the "mechanical" calculating and computing done by the electronic brains (!) of that time was much much faster than ever humans could imagine. The "Giant Brain" (as the press called it) ENIAC from 1946 could do unbelievable 35 divisions or square roots per second, and unimaginable 357 multiplications per second! This enormous speed of elementary, but for a human quite time-consuming mathematical operations was the highest peek of computing productivity anybody could conceive, as thousands and thousands computers would have to be employed to do the same amount of calculations in the same amount of time as the giant electronic brain(s).

It could be noted that this thrilling speed of something humans see as very complicated operations (multiplication, division, square root) raised the eyes of everybody towards the unfathomable heights of "intelligence" that were in front of us. Because, if that electronic brain can do so complicated operations so extremely fast, he must be able to be thought (instructed, programmed) to be highly intelligent! We just need to find how to programme intelligence into the giant brain... and soon: it will help us solve all of our problems / it will take over the world (choose by preference).

Therefore it is quite logical that with such an enormous difference of the calculation speed between the electronic computer and the human computer the productivity was fully focused on the electronic "brain". Any (well almost any) amount of human labour to prepare the calculation sequence, the way the computer will perform the process, the algorithm, and the data necessary - was a very small general effort in comparison with the amount of human computer effort saved by the use of the computing equipment.

However, it is interesting to note that the balance of productivity is always important. In 1948. a small stored programme memory was added to the ENIAC computer. This modification made the computer to work much slower, and disabled the possibility of use of parallelism in the processor. Although this modification reduced the speed of the computer by a huge factor of six and in the same time eliminated the ability of parallel computation, it did reduced the reprogramming time from days to hours. The speed of the computer was still high enough to be finally I/O bound, and the productivity was enhanced: the change in speed of programming was considered well worth the loss of performance in execution.

An important fact which has to be mentioned from the historical perspective is that the memories of the early (electronic) computers - both the data and the programme storages, independent of the architecture - were quite small, and their size never surpassed anything a human could not analyse on paper in a reasonable amount of time. Therefore a convenient method of getting rid of algorithmic, data or human errors in the calculations on a computer was the so called "core dump", meaning a full listing of the data in all computer memory locations. This would then be used very effectively by the humans to understand what went wrong. And just to mention an important historical fact: the first electronic computer programmers were women - the human computers employed during the war.

II.2 The "Middle Ages" of Computers

Very soon it was realised that instructing the computers on the basic level of wires, bits and switches is not the most productive way of translating algorithms into actual computing processes. More freedom of expression was needed for the programmers, more autonomy was needed from the computers.

The focus of productivity partly shifted from the previous era, as more and more computers, larger and larger computer memories and faster and faster processors began to be available. Now the challenge started being how to organise all of our ideas (and we just mentioned how high our hopes were flying) and somehow transfer them into something the computer would understand, as the performance of the computers was drastically raising even inside the same decade. Suddenly the computers started being "hungry" for programmes and new data, as most of the simple algorithms and small data sets would be "devoured" by the computing speed.

So, to cope with this change of the focus of productivity, as now the computers were already so stable (hardware-wise) and so fast that we, humans, started lagging behind with our job of programming, we had to invent a new way to organise our process of instructing the computers. That led to the invention of autocoders, assemblers and higher level programming languages.

However the way the computers worked, i.e. the way those very basic operations are organised in streams of operators and operands, was very influential on the development of the emerging "programming languages". Actually, the computers themselves were initially constructed to do sequential mathematical algorithms, the

same way a human computer would do them. They were never in construction then intended to process and render movable pictures, to translate natural languages, or to drive a car. Although between the first few computer programming languages ever invented we see several major families of completely different approaches, as in the families (in the broadest sense) started by Assemblers, Fortran, Lisp, APL and Cobol, hardware was primarily (and still is, unfortunatelly) developed based on the same philosophy which, out of necessity, produced the first electronic computers.

So consequently, most of the programming languages were/are based on the elementary serial calculating engine principles, as to be effectively executed on hardware. This was a combined consequence of the common computer architecture and the fact that vast majority of programmers, being thought about such computer internal working and architecture, very easily accepted just those languages, and not the other computer-linguistical streams like APL or Lisp.

II.3 Where Are We Now?

In the meantime our "necessities" have drastically changed. We do not have any more relatively simple needs that could be explained to the computer in several hundreds, thousands or even tens of thousands of elementary steps. We now, more and more, want computers to perform well in a very vast field of extremely diverse data and information processing domains. Suddenly, historically speaking, we have the wish, which we perceive as a need, to process huge amounts of data, to process extremely complicated algorithms, and we wish we would have computers which could process zillions of over-simple operations, common to our present equipment as they are inherited from the early history of computing equipment. Not many different hardware approaches did we ever try, and most of the alternatives we do not use. And, furthermore, the same type of equipment we also wish to drive cars, move robots, keep our doctor informed about our blood pressure, get the lights properly running in the theatre, and allow us to chat through some book of faces.

And naturally, consequently our focus of the notion of productivity changes again. This time we have an enormous amount of diverse computers, all based on processing very simple operations on any kind of operands presented to them as data, and most of them organised as scalar serial processors. And we have an extended set of programming languages almost all of the same kind, the basic historical computer-execution oriented kind. And even more, most of us use just the linguistically worst languages, i.e. those at the lowest machine level, as e.g. Fortran, C, C++, Java... etc. Though some of us love some of them, others some others, still others keep their faith in Lisp or APL families, we are all gravely and thoroughly aware of the fact that programming anything for the present day data communicating and processing environment is extremely tedious and with results which can not be predicted to be as we wished, wanted or needed.

The productivity is getting lower and lower. The path between the idea and the final realisation, the final answers or possibility of application of the computer is very very long. We cannot cope any more with the complexity which emerged.

So the focus of productivity in present day and, as much as we may predict the future, in the future shall be on the notion that computers, as being more and more inseparable part of our whole society and civilization, have to be user friendly to any educational level of potential users. Therefore the notion of High Productivity in this sense encompasses the productivity of any human necessitating help from information processing and computing equipment, whatever their needs be.

III. Humans and Computers

So we stand here, in front of unknown powers and possibilities, in front of the Giant Brain encompassing presently billions of computers of all kinds, intertwined into a twisty turbulent network of slow, medium, fast, very-fast, ultra-fast connections, and we would like to actually use it. As a personal assistant, as a scientific collaborator, as a companion in times of leisure, as a librarian and as a library, as a preserver and a collector, as a future telling machine, as a trusty banker and as an "intelligence enhancer". And not to forget, we would like this Great Brain also to drive our cars, to play waiters in bars and lounges, and restaurants, naturally, and also to produce cars, to take care of our health, and, most importantly, to entertain us.

There is a very interesting aspect of the interrelationship between Humans and Computers. Though this is not the main thematic of this article, it is a very important aspect of our present day overall civilisation development:

Imagine in one moment a huge electromagnetic storm comes onto our little planet Earth. And suddenly the internet is down. Completely. Could we succeed to live as we knew just twenty years ago? Could we survive with the non-connected computing technology?

Imagine in one moment a huge electromagnetic storm comes onto our little planet Earth. And suddenly no one computer works. All the buzzing computing equipment is dead still. Could we succeed to live as we knew just seventy years ago? There are still quite many between us who lived already seventy years ago. Could our present day civilisation survive the shock of being thrown back just half a century? (We would still have the knowledge!)

Would that not be good stories for a science-fiction horror film? Humans Without Computers!

Well, to continue now, do not worry and do not imagine that any of the above storms happened. The major problem we, as Humans, actually have with Computers from the very begin of their development is that the communication possibilities between a human with an idea and a computer into which it could be implement are so primitive that not many of us either can grasp all the necessary prerequisites, or have enough time to do it, or are interested at all to tackle such a complex field of communication. Even if we have strong intentions and enormous sitting stamina the job of explaining any idea to modern day computers, specially if we need more than one processor, or more than one computer, is a risky job, as a lot of unpredictable problems related to any level of modern computer realisation (hadware and appropriate accompanying software) can suddenly emerge and throw us into a next frenzy, undefined time frame of "development". Most of what we euphemistically call *development* of an application (of ideas onto computers) is actually headbanging debugging and search for solutions of practical implementation problems in a dimly lit space of extremely high complexity, meandering through the vast and confusing ocean of individual low level statements.

III.1 Human Civilisation and the Language

The Language is the prime human tool for building a common model of the Universe. Cybernetically speaking, the Language is a huge fractal dynamic system, i.e. machine, whose sub-machines are smaller fractal machines consisting of a series which goes through dialects, sociolects down to idiolects. On each fractal level the language is defining the model of the world, from the individual, with his knowledges and prejudices, through a specific group, society, field of interest, with its specific terms and phrases and implicities, up to the national Language, with its approach towards life, its epistemology, paradigms, hopes, fears and spirituality. Not forgetting the highest level of human common, some and all languages encompassing, model of the environment in which each and every Human Being lives, as for example the notions of birth and death, the notions of Sun and Moon, leaves, trees, water and vapour, love, heroism, serenity, knowledge etc. etc. From that level down the facet machines, sub-languages which work inside a Language may, as we get towards the individual, be getting more awkward, slangish, highly distorted, very narow-minded, and logically must be entailing smaller amount of active and passive elements. The "meaning" of something is always seen and communicated through the informational structural position of notions transferred into speech by organisation into words and phrases and enveloped by idiomatic linguistical rules into a specific text, intended to have the possibility to "rearrange" the informational structural position of notions in the other person, group, society... One human knows something, many humans know much, all humans know all (what the Human race as a collective knows).

In other words, a natural Language is a very flexible web of interrelations between notions up to the level of text expression. That way it becomes a common model of all that a human and his human civilisation perceives in the world around. Through that it is obvious that all inter-human communication, to be able to express any reference to things not here and now, i.e. to use the frame of reference of a common model of the World and influence the change of particular aspects of the model in communication co-actors, has to be linguistically based. Even more, due to that baseness of Language in our perception of the environment, most of our internal thoughts are very often in the form of words and sentences of a Language which is internal enough for us (mother tongue or tongues or well known, interiorised, other languages). The Language, constantly changing and adapting itself, as a whole defines this active frame of reference through which we inspect our environment and communicate about it.

Are you gay? In a melancholic way? I hope you are not just sad! You know, presently most computers are women!

These few sentences probably exemplify the above text quite well. They seem strange in present day English, and they, to be properly understood (as they were intended to be by the speaker/writer) have to be read/heard within an appropriate *context*. How much information did you get by reading those sentences literaly? For this example the understood meaning during the Second World War America would be quite cheering up for a mathematically versed unemployed woman, staring through a window in a gay, but melancholic way.

III.2 Programming Languages

As already mentioned, the development of programming languages was a natural extension of what the computers could execute towards the perceived needs of what we would like to "compute". It was a huge step forward in the sense of productivity as opposed to the defining and rewiring of specific wires connecting computer subunits, or entering individual bits of instructions into the machine. The higher level syntax, rigid and formal, as to be able to be easily and unambiguously translated into elementary machine instructions, and the use of words similar to some words of natural languages (but absolutely strictly unambiguously defined) made instructing electronic computers, programming them to do something according to some explicitly in the programming language described algorithm, much easier and more convenient. However, the main benefit of using programming languages with a reasonable syntactic and sematic expressibility was (and is), naturally, the possibility to use a higher level of abstraction, and therefore to think easier and more about the algorithmic translation of ideas then the actual low level hardware execution prerequisites.

Therefore the introduction of formal computer programming languages meant a higher level of abstraction, enabled portability and generic algorithmic formalisation, and all that with much less steep learning curves.

III.3 Which Programming Language?

"When tackling a complex new problem first develop a formal language specifically oriented towards that problem, the problem is then much easier to solve" (paraphrasing William M. Waite), or use an existing formal language which is already developed for the necessary type of processing.

Quite interestingly, although it is quite simple to develop and implement a specific field attuned formal language, using standardised tools, and although the productivity of writing in such a specialised language is much higher than in generic formal languages, this very valuable approach seems to be all but forgotten. Even the usage of different already developed and implemented formal languages which are specifically adapted for certain areas of problems, and can yield really much in productivity inside their specialisation, is presently not common. It seems that everybody wants to do everything in just the one (or very few) formal language(s) well known (and actually the popularity of those most popular - C and its family - is a consequence of the C compiler being part of all standard UNIX distributions up until recently, and not a merit per sui generi). Does it matter at all how hard certain things are to program, or how complex certain algorithms come out in those over-popular languages?

Actually almost all our computer linguistic efforts are generally oriented towards making a formal language which could be easy enough for a human to understand and learn, and still simple enough to be easily and efficiently translated into the scalar serial frame of mind of the prevailing computer architecture. They all somehow tend towards the ideal of mathematical notation, algorithmic notation, but also towards the ideal of a natural way of expressing. However, the mathematical and algorithmic notation used in inter-human communication is still too much away towards the natural languages to be realistically regarded as a formal language. These systems of notation, developed ages ago and constantly

enhanced, were never meant to be self-standing in communication, but were and are an "enhancement", a linguistic field-specific addition to the natural language, primarily in the area of the appropriate sociolects. Many other such notations as the mathematical do exist, in a wide variety of domains, with more or less internal formality, consistency and information content inter-fixation - which is the highest in mathematics. All of them are used in inter-human communication only inside natural language texts!

IV. Still Computers? Actually No... Ordinators!

We have seen that the major focus of our use of electronic computing machines has fully changed, from the time of being equipment to compute numerical results of mathematical algorithms to the present time of being equipment to process complex information structures and flows based on algorithms from diverse fields of sciences, arts and communications.

Though we still call our machines computers, actually they are much better described by the French word "ordinateur" (equally in e.g. Spanish, Italian...).

Or perhaps we shall keep this name, *the Ordinator*, for the next generation of machines which will actually process information and not data, and which will be completely different in their architectures, as to allow productive processing based on natural-like languages, where many algorithms will be automatically chosen and optimised based on the knowledge of the context on the lowest levels of hardware and software. And with which humans will communicate in a consistent and understandable way.

IV.1 The Hardware/Programming Languages Bottleneck

Once upon a time we had Simple (as in "then constructable") Hardware architecturally serial scalar and for that kind of hardware we developed Simple formal Languages, expanding them as necessary to cope with as much low level programming as possible in a reasonably "high" formal language environment.

70 years later we extensively use Simple formal Languages - whose development started once upon a time conceptually for serial scalar hardware architectures - to make programmes to be executed on Very complicated Simple Hardware, the direct descendants of hardware constructed once upon a time.

It seems that we got stuck in a vicious circle: The development of new hardware approaches, which would be radically different form present day architectures is not regarded as viable, as there is a lot of "software" which shall be directly compilable/executable on any new or old computer system. Therefore the programming languages keep the possible further hardware and computer architectures development in check. The development of new linguistical approaches, which would be radically different in the sense of much higher, towards the human oriented, expression power, is hampered by the necessity to be implemented using existing programming languages and programming paradigms and for them appropriate hardware. Therefore the hardware keeps possible the development of future alternate and novel human-computer communication languages in check by being very to extremely unfit for effective execution of such communication/programming paradigms.

IV.2 Drinking Water From a Glass

Let us, for a moment, pretend to be very obedient, and execute the following instructions: There is a glass of water on the table. If there is water in the glass do the following. Take a toothpick. With the toothpick acquire a drop of water from the glass. Put the toothpick between your lips. Swallow the drop of water. Return the toothpick. Execute again from the second instruction sentence.

Well, now do it 2 billion times a second, instruction by instruction, drop by drop. What a slow and tedious way to drink a glass of water! So what shall I do? Use a fork! So I will process four drops at once! - Still very slow. So let us try to use for example a hundred thousand toothpicks, or even forks! That shall do the trick! Well, it would... if the glass would not have been to small for so many forks, and if somehow we could solve that terrible problem of COORDINATION of 100.000 hands, some left, some right! And even with the two of them often the one does not know what the other does. (Well, the whole exercise with 100.000 forks picking up the drops 2 billion times per second has actually no sense, as the amount of water drops in a glass of water is only around 5000. So we did a monumentous overkill!)

But it would have been fair (and much much more productive) that somebody have said to you, instead of giving you this silly instructions text, to drink the water from the glass on the table. It would have been much easier for both of you. He could have said it in a simple and to him also easy understandable way, and you could have ignored the toothpick and the fork and used your hand to pick the glass and drink the water from it. This is the difference in productivity based on just the use of proper high level contextfull language (context here, inter alias, being the knowledge of the notions table, glass and drink on both sides of the communication channel).

(Any similarity to present day usage and construction of computers is purely coincident.)

V. Natural vs. Formal Languages

Every Language is in its basis Formal (otherwise we could not understand each other). However this formalness of the natural languages is not specified outside itself, as with strictly formal languages, but is inbuilt in the very essence of the language and the corresponding model of the World. Therefore it is flexible, and the tensions of these flexibilities spread over the society using the language pull certain notions, grammatical rules, phrases, words and pronounciations to new positions in the interrelations network, giving them new meanings or new expression forms.

However all this is possible because natural languages have Context which is mostly implicit in the use of words and their sequences, i.e. phrases, sentences, texts... Therefore it is completely appropriate to use the same lexical word in conveying different meanings, the so called homonyms, as the context allows (if there is enough of it) proper "decoding" of the meaning. Homonyms are two different words with different meanings but which look and sound the same. Something like *cool* in "This jacket is really cool" - i.e. 'Wow'; and "This water is really cool", i.e. 'Brrrr'. Programming languages do not have implicit knowledge of the context and therefore no possibility for usage of homonyms. Contrary, if using information context of what the objects (data) of a specific verb are, what they represent

- that is directly usable to properly define the necessary algorithms to process the data according to the appropriate homonym.

Another important element of all natural languages are synonyms (although theoretically there is always the language efficiency principle acting, which on a longer run disallows complete, full synonyms - no two natural language words have exactly the same meaning).

Formal languages can not describe novel grammatical and semantic rules of themselves. They are never self-referential. Natural languages are a-priory self-referential, as the only possible definition of the language is inherently inside itself, i.e. inside the fractal knowledge of all speakers of that language.

There is also a third kind of languages, in between the formality and rigidity of formal languages, and the naturality and flexibility of natural languages, let's call this language family Natural-Like Languages. These languages, though formally defined in their basis, enable the expansion of their own linguistical rules, of the grammar, of the lexic and of the semantics and context-environment handling. The axiomatic language of mathematics is such a system, which can be expanded and contracted in grammar, in lexic and in semantics, by the proper use of already defined phrases (constructs). Unfortunately the consistent possibility to enhance and change the grammatical rules of itself is not existing in strict formal languages, as are our programming languages. This consequently also leads to the impossibility of algorithms written in "programming" languages to be meaningfully linguistically combined.

VI. An Approach Towards High Productivity Computing

It is obvious that presently we have a huge heap of more and more complex problems to solve if we want to continue our expansionist usage of *data* processing and communication equipment. It is also obvious that we actually need *information* processing and communication equipment, with a much higher productivity of the human-ordinator coordination.

VI.1 Present State of Affairs

When "programming" computers we talk really a lot in a language which can express very little with its grammar and lexis, so we have to talk so much to be able to define at all any even slightly complicated processes and define them well. In other words, we use languages which say very little with a lot of words. Using a language which says a lot with a few words it is possible to understandably express something not trivial in a very short text. Only than are we able to use a lot of words to say really much.

Yet we succeeded to develop enormous amounts of algorithms and by that gained huge quantities (and qualities) of knowledge how to implement our wishes in a formal way. This knowledge, gained in a very hard way, by programming those computing engines which know how to execute only a very few very elemental mathematical, logical and organisational operations, is extremely worthy. But we have to regard it as knowledge and experience, and we shall be very careful by taking those algorithms, those programmes, those applications literaly, as they are now, into the future. By regarding these historical developments in computer science primarily as knowledge founded on experience we are freed to use the same ideas we used to implement on classical computers in the future on completely novel ordinator architectures, expressing them in completely novel natural-like, human language similar language(s).

Much can be said about the rationale of the above-said. We presently use layer upon layer upon layer of extremely complex sequences of extremely simple "instructions" in those "modern" programming languages we use. This results in high levels of unnecessary processing, from the level of the operating system, to the level of the highest layers of user-land. We have scalar serial processors and try to connect them in clusters, grids and clouds, without a general linguistic notion of how to program uniformly those (heterogenous) processing elements. Some of our algorithms may work properly with 32 processing cores, but do not gain anything, or even lose, if we use 64 processors. How do we expand such algorithms to work on thousands of processors? We talk a lot about information processing (even the whole area of human effort is called Information and Communication Technologies), but most of the numbers (specifically on the processing level) we process are pure non-tagged data. Enormous amounts of bits in bytes representing something which could be either enormously overgrown so called Applications, or it could be pictures, films, cardiograms, parts of a Beethoven symphony, in miriads of formats, or anything else, as a matter of fact. The processing unit has absolutely no idea what the data processed represents. The storage unit neither. Nor do we, if we lose the "directory list". We almost know how to program single scalar serial processors, how do we do it when we want to program and coordinate hundreds, thousands, millions? ...and even more, we shall be productive when doing it.

Unfortunately the above short list of just some problems to be solved is far from being exhaustive, as we approach the era of peta-, exa-, zetta-, yotta-. (Yotta is the approximate estimated amount of stars in the presently Observable Universe - that is all stars in all galaxies we could observe by the most modern methods of observation. Our own galaxy, the Milky Way, consists of less than half a Tera of stars.)

VI.2 Data vs. Information

"10.77032961 10.44030651 10.19803903 10.04987562 10.04987562 10.1803903 10.44030651 10.77032961
10.81665383 10.29563014 10.29563014 10.81665383 10.63014581 10.63014581 10.63014581 10.63014581
10.81665383 10.81665383 10.29563014 10.29563014 10.77032961 10.77032961 10.44030651 10.44030651
10.19803903 10.19803903 10.04987562 10.04987562 10.04987562 10.04987562 10.19803903 10.19803903
10.44030651 10.44030651 10.77032961 10.77032961 10.29563014 10.29563014 10.81665383 10.81665383
10.63014581 10.63014581 10.63014581 10.63014581 10.81665383 10.29563014 10.29563014 10.81665383
10.77032961 10.44030651 10.19803903 10.04987562 10.04987562 10.1803903 10.44030651 10.77032961"

Cited here is some data. Obviously a list of numbers, all in an open interval between 10 and 11. This is quite apparent.

However, not much can be done with the above list of data. It is actually meaningless as such. It just happens that I personally know what those numbers represent, as I did generate them, but then, on the other hand, as I did not write any explanation, this list of numbers actually can have any "meaning" anybody gives it, as long as that "meaning" preserves the consistency of the internal data relationships. Or not... Perhaps it is just individual numbers which have no internal relationship at all inside the list? Or should it perhaps be a table, and not a list?

This example shows quite obviously that Information is not Data only. That no meaningful information can be extracted from the presented list without deep investigation, if even then. If we would

use these data as information(s) we have to have a context in which the data has some meaning, some sense. Therefore to talk about information it is necessary to have a context. For the area of data/information processing we could define the notion of information as being a combination of data and metadata - the value and the context, or, in the parlance of linguistics, the significator and the significant, the expression and the content. Though it may seem confusing, we can take as a premise that, for our purposes, the data is actually the expression and the metadata the content, as the data actually expresses a specific value of the same type of content. Therefore we could write Information: Data + Metadata.

It could be suspected, by not knowing the whole truth, that mathematical notation does not involve context. Actually it always does, in a crude way similar (but on a much higher model level) to the (formal) use of I-, J- and K- names in Fortran, which on the level of the (unspoken) language inherent model have a presuposed "context" of being integers, if not explicitly defined as something else by some sentence of the programme. The meanings of mathematical notation (the same as any other, e.g. musical notes) are deeply rooted in the human language and the model of the World, and directly contextually connected with the enveloping linguistic expressions. Humans never process data - only information. The context of this information is always deeply rooted in the perception of the environment, independent of the level of "education" or "knowledge" or "intelligence". And natural languages, consequently (and presequently) handle only Information - even the bees when communicating the location of usable flowers!

VI.3 An Approach Towards High Productivity Computing

Finally, given here are some indications of paths which shall be taken, or at least explored, and which can lead us towards a much higher level of productivity in our cooperation with Computers (or Ordinators of the future).

Firstly we shall use Information, being here defined as context-aware Data and Meta-Data, and not use Data alone. This is the prerequisite change on the paradigmatic level which enables the development of natural-like languages. Further highly important features of natural languages are that they are "eclectic", meaning that different styles, different synonyms and even different grammatical rules on different semantical fields can be seamlessly integrated into the Language. Well, otherwise we could not communicate in the society at all! The use of a contextful language, dependent on information and not data alone, enables the very important and productive usage of homonyms (a simple example would be the usage of a multiplication operator on two complex numbers, where the type of the numerical content - polar, cartesian or dimensionless - is known to the processor, so that actually we have the same word doing mathematically quite different operations, where the results may, as in this case, or may not be inter-compatible, as when multiplying a character by 0 or 1, taking it none times or one time). And, very importantly (and extremely easily implementable) - for the benefit of Humans the language has to have synonyms.

To raise the level of productivity of the human-computer cooperation, it is essential to try to level up the basic Ordinator (Computer) language understanding towards the level of Humans (and not vice versa!). This will enable to significantly widen the approach towards construction of basic hardware processing units and their architectures. As already stated, the present formal languages interlocked with present hardware architectures do not enable efficient implementations of natural-like languages. Naturally, as they are Turing-complete, it is conceivable and attainable to implement this kind of language understanding using present day languages and methods. This is very relevant for the interoperability of approaches towards High Productivity Computing, as the same linguistic system can be implemented in software, as a Virtual Executor, on any convenient spread of present day computing architectures, as well as on specially developed hardware and computer architectures. Naturally, as already explained, the software implementation is sub-optimal per se, and only full development of natural-like human-computer communication and interaction languages implemented in the lowest levels of hardware, microware and firmware will enable a full growth towards the aim of keeping the productivity of information processing and computing as high and as balanced as possible between the two actors in the process - the Human and the Ordinator (computer).

VI.4 An Experiment in High Productivity Computing - Virtue

At the Rudjer Boshkovich Institute in Zagreb we started tackling these problems systematically several years ago, and an investigation into basic principles of computer programming, as well as the principles by which we serialize the inherently extremely massively parallel universe around us into serial algorithms was performed - and a new approach taken.

The result of this approach is Virtue - the Virtual Interactive Resource-Tasking Universal Environment, an experimental implementation of an approach towards High Productivity Computing.

Imagine a mathematically simple and effective visualisation - a four-dimensional hypersphere to be rendered as simple asterisks showing all the dots inside the sphere's radius, and spaces outside, the third and fourth dimension to be shown as a sequence of two-dimensional slices. By just looking at such a very elementary visualisation the basic structure of the hypersphere can easily be deduced. Now find a C, C++ or Java programmer and ask him to make a short programme to show such a sphere to you... And now, after they have shown you this four-dimensional sphere, ask them to show to you how it would look like in 5 or 6 dimensions... Or, better, do not ask them to do any of that for you in such a language, as it would take quite a lengthy time even for the best. (A non-optimised solution which we prototyped in C for 4 dimensions has more then 130 lines of source code - non-paralelised! An optimized or parallelised version would take much longer to develop.) And it is a kind of "one-execution is enough" request and programme.

The definition of a new Virtue verb "sphere" which solves the above problem for any number of dimensions up to 8 [taking three scalar numerical objects, expressed as a real number for the radius, and two n-dimensional numbers (real - 1d, complex - 2d, quaternion - 3d or 4d, or octonion - 5d, 6d, 7d or 8d) for the centre and the space-size] is this sentence: TRIADIC SPACE CENTRE MAGNITUDE GREATER '*' MULTIPLY; OPERATOR @sphere SET. For example: 15 15i15j15k15 31i31j31k31 sphere.

What it says is that you want to have an asterisks wherever the radius is greater than the magnitude of the centered space elements;

and otherwise a blank. The verb SPACE (synonymous to INTERVAL) with a scalar numerical object makes a space of indices from 1 (or 1i1, 1i1j1, 1i1j1k1, ...) up to the specified last indices, i.e. the indicated size of each dimension. The word CENTRE is a simple synonym for the verb SUBTRACT (scalar subtract of n from an index array actually centres it around that n).

Being based on such principles, Virtue is a language which proposes a different approach, by keeping the inherent parallel structure of natural algorithms, and doing the parallel processing by itself, if it is algorithmically possible and feasible. Virtue is a syntactically very simple, yet semantically extremely complex language, offering, inter alias: consistent application of any operation on any logically usable combination of data-types, no "reserved words" and almost no "reserved interpunctions", synonyms, automation of memoisation, multiple word contexts (and therefore homonyms), combined data types of anything Virtue supports (e.g. functions, symbol names, scalars, multidimensional sub-structured spaces etc.), stochastic processing, multivalued and multidimensional logic operations, multidimensional sub-structured file access structures, continuations etc., SIMD, MIMD, SISD and MISD programming models and furthermore also allows for the changes of its own grammar.

Hierarchically, by defining complex meanings for Virtue words (or, differently said, named functions defined as operators), as e.g., building on the previous example, by defining the "meanings" of 'small' 'big' and 'please' thus: NILADIC 5 5i5 11i11; OPERATOR @small SET. NILADIC 30 30i30j30k30 61i61j61k61; OPERATOR @big SET. NILADIC; OPERATOR @please SET., high level simple natural-language like expressions can be used: "small sphere please." or "please big sphere.". Being interactive the Virtue Environment enables easy and understandable development and debugging (including tracing, stepping, intervention, editing...). Used in batch mode, a programme written in Virtue may be used for computing input to other programmes, as for example PowRay for visualisations, or the Virtue environment may be used as a pipe-through between any existing programmes/applications.

Therefore, due to this semantic richness and grammatical simpleness, in Virtue, for example, the text of the algorithm for Conway's "Game of Life" necessitates only 12 language tokens (7 words, 14 numbers in 3 vectors and 2 delimiters) in one sentence: MONADIC (1 1 1 1 0.5 1 1 1 1) [3 3] MONADIC RAVEL SUM (2.5 3 3.5) IDENTICAL ANY; STENCIL;. (The STENCIL is a stenciling operator, synonymous to MASK. The array edge behaviour can easily be modified by optional modifier words REFLECT or WRAP. These operators will work on any-dimensional spaces.) This sentence (algorithm/programme) will work for any size of a "Game of Life" board, up to the size of the underlying hardware memory, and automatically using, if feasible, parallel processing (actually a higher level form of SIMD in this case).

The computer implementation of Virtue is presently in Alpha 0.7 state, and parallelisation is implemented on single-image systems. The experimental implementation is constantly parallely developed and tested on a very wide range of different computers, ranging from the mid-1980-ies Sun3 (16MHz/16MiB and 20MHz/24MiB) workstations up to modern day blades, with various operating systems and their generations (SunOS, Solaris, NetBSD, FreeBSD, Linux, MacOS/X, UWIN, Cygwin...), different compilers and compiler generations, various processors (M68k, MIPS, SPARC, PowerPC, AMD, Intel), on 32 and 64 bit architectures, 32 and 64 bit floating-point, in single-processor and SMD/NUMA multi-processor, multi-threading and multi-core computers. Such a wide range of computers, both historically and speed-wise, for the experimental Virtue implementations allows for development of a very easily adaptable system, and the behaviour of the old Sun3 systems shows that even on them the execution is fast for the amount of data which can be represented in the memory of those computers (as an example, executing the classical double-recursive fibonacci algorithm, a full list of all the *first 1500* Fibonacci series numbers will be produced on a 1986 20MHz Sun3, using the inbuilt Virtue memoisation word RESULT, in only *27.827s*, whereas the next time the same request is entered the results will come in just 4.461s - using the same basic algorithm in Virtue, a non-memoised recursive calculation of the (just) 32^{nd} Fibonacci series number on a modern day SunFire X4240 takes full *26.012s*!).

The internal speed measurements which the Virtue Executor has have provided us with quite a lot of important data on the behaviour of different computer systems and different processors, so an investigation into the "speed of a computer" is presently being performed, with some results to be presented soon.

Shortly presented in this subsection is an experiment in computer implementation of the necessities and possibilities of High Productivity Computing. Further development of this idea allows for definition of a more syntactically rich very high level human oriented machine interaction language, which, combined with additional artificial intelligence components, appropriate ergonomic human presentation/sensory interfaces and with the integration of user style association memory, we sincerely hope can help the future development of Computer Science and Usage Practice.

VII. Focus Group High Productivity Computing

Under the umbrella of Working Group 1 ("State of the art and continuous learning in Ultra Scale Computing Systems") of the NESUS Action (COST IC1305 - "Network for Sustainable Ultrascale Computing") the "Focus Group High Productivity Computing" (FG HProC) was established. The main aim of this Focus Group is to explore the possibilities of an integrative approach which would allow a significant shortening of the time lapse between the human ideas or needs and computer implementation solutions. As already quite thoroughly explained in this article, the major idea driving this Focus Group and its (present and future) work is that computers, as being more and more inseparable part of our whole society and civilization, have to be user friendly to any educational level of potential users, and the notion of High Productivity in this sense encompasses the productivity of any human necessitating help from information processing and computing equipment, whatever their needs be. This is the idealistic aim towards which this Focus Group will try to steer itself.

To achieve this global aiming, the Focus Group will focus on two major aspects of computer science: the past (and present) and the future. To learn from history: a comparative exploration of computer history and present day tools, methodologies, languages, algorithms and hardware in view of the information processing and computing needs and necessities perceived now and, as much as possible, projected into the future. Some of the preliminary investigation results are given in this article. An extremely important aspect of this is historical research into avenues of computer science taken, but not pursued. A huge amount of great ideas is actually

almost forgotten, ideas that can be very useful today, and for which in the time they were invented and thought out there was not a feasible possibility to actually be realized in their full potential. This exploration will be a very valuable addition to the NESUS WG1, regarding the State of the Art, as well as to the general computer history, as observed from the technical and technological side.

To be able to define the necessities of high productivity in the sense of the aim of this Focus Group, another objective is important, the exploration of the concept of High Productivity Computing, or, maybe better to say, High Productivity Information Processing and Computing. The objective of this Focus Group is to explore the development of necessities in the area of high productivity through the history of computer science to present day, and produce knowledgeable reports on possible future shifts of the productivity focus. Prime goals are the envisioned resulting knowledge gained from extended study, and a strict definition of the concept of High Productivity in Information and Computing Sciences.

The central objective of the Focus Group High Productivity Computing is the development, definition and standardization of a universal (linguistic) environment from the level of (new) hardware up to the level of algorithmic expression of complex algorithms involving a wide spectrum of available information processing and computing equipment, and based on the algorithmic and linguistic knowledge gained so far by our civilisation. Based on these the goal is to actually be able to realise a prototype of a High Productivity Information Processing and Computing Infrastructure by facilitating cooperation of different interested partners through the FET and other H2020 European Union science founding calls.

VIII. The End:

– Use courage to "Boldly go where no other computer scientist has gone before"! –

References

[1] C. Backus, "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs.," *Communications of the ACM*, vol. 21, no. 8, pp. 613-641, August 1978.

[2] Y. Bar-Hillel, *Language and Information*, Addison Wesley, Reading, MA; London, U.K, 1964.

[3] D. R. Hofstadter, *Goedel, Escher, Bach: An Eternal Golden Braid*, The Harwester Press, G.B, 1979.

[4] L. Floridi, "Is Semantic Information Meaningful Data?," *Philosophy and Phenomenological Research*, vol. LXX, no. 2, pp. 350-370, March 2005.

[5] G. Lerner, "The Necessity of History," in *Why History Matters: Life and Thought*, New York, NY, USA, 1997.

[6] J. S. Light, "When Computers Were Women," *Technology and Culture*, vol. 40, no. 3, pp. 455-483, July 1999.

[7] E. Lusk, "Languages for High-Productivity Computing: The DARPA HPCS Language Project," *Parallel Processing Letters*, vol. 17, no. 1, pp. 89-102, 2007.

[8] S. Marin, M. Ristic and Z. Sojat, "An Implementation of a Novel Method for Concurrent Process Control in Robot Programming," *Third International Symposium on Robotics and Manufacturing: Research, Education and Application*, ISRAM '90, Burnaby, BC/CA, 1990.

[9] J. Mrsic-Flogel, D. M. Reynolds, Z. Sojat, M. Bianchessi and S. Sala, *Data Communication*, International Patent, US Ref. No: 7565210, 2009.

[10] K. Skala and Z. Sojat, "Towards a Grid Applicable Parallel Architecture Machine," *Computational Science - ICCS 2004, 4th International Conference* KrakĂşw, Poland, 2004.

[11] Z. Sojat, "Principles of Selforganizing Learning and Their Application on a Computor Program," *Pan Arab Science Fair*, Kairo, Egypt, 1976.

[12] Z. Sojat, "Pri la problemo de lingvsignkontinudiskretigeco kaj la sia masxinrealigo," *10e Congres International de Cybernetique*, Namur, Belgium, 1983.

[13] Z. Sojat, "An Approach to an Active Grammar of (Non-Human) Languages," *27. Linguistisches Kolloquium*, Muenster, Germany, 1992.

[14] Z. Sojat, J. Mrsic-Floegel and D. Moritz, "The Facets of an Information," *II. Orwelian Symposium*, Karlovy Vary, CZ, 1994.

[15] Z. Sojat, T. Cosic and K. Skala, "Virtue - A different approach to human/computer interaction," *Information and Communication Technology, Electronics and Microelectronics (MIPRO); 37th International Convention on*, Opatija, Croatia, 2014.

[16] A. N. Whitehead and B. Russell, *Principia Mathematica*, Cambridge University Press, first published 1910.

[17] L. Wittgenstein, *Philosophical Investigations*, Blackwell, Oxford, U.K, 1953.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'.